Abstract

This document describes a collection of mechanisms that allow HTTP servers to cryptographically bind authentication tokens (such as cookies and OAuth tokens) to a TLS [RFC5246] connection.

We describe both _first-party_ as well as _federated_ scenarios. In a first-party scenario, an HTTP server issues a security token (such as a cookie) to a client, and expects the client to send the security token back to the server at a later time in order to authenticate. Binding the token to the TLS connection between client and server protects the security token from theft, and ensures that the security token can only be used by the client that it was issued to.

Federated token bindings, on the other hand, allow servers to cryptographically bind security tokens to a TLS [RFC5246] connection that the client has with a _different_ server than the one issuing the token.

This Internet-Draft is a companion document to The Token Binding Protocol [TBPROTO]
1. Introduction

The Token Binding Protocol [TBPROTO] defines a Token Binding ID for a TLS connection between a client and a server. The Token Binding ID of a TLS connection is related to a private key that the client proves possession of to the server, and is long-lived (i.e., subsequent TLS connections between the same client and server have the same Token Binding ID). When issuing a security token (e.g. an HTTP cookie or an OAuth token) to a client, the server can include the Token Binding ID in the token, thus cryptographically binding the token to TLS connections between that particular client and server, and inoculating the token against theft by attackers.
While the Token Binding Protocol [TBPROTO] defines a message format for establishing a Token Binding ID, it doesn’t specify how this message is embedded in higher-level protocols. The purpose of this specification is to define how TokenBindingMessages are embedded in HTTP (both versions 1.1 [RFC2616] and 2 [I-D.ietf-httpbis-http2]). Note that TokenBindingMessages are only defined if the underlying transport uses TLS. This means that Token Binding over HTTP is only defined when the HTTP protocol is layered on top of TLS (commonly referred to as HTTPS).

HTTP clients establish a Token Binding ID with a server by including a special HTTP header in HTTP requests. The HTTP header value is a TokenBindingMessage.

TokenBindingMessages allow clients to establish multiple Token Binding IDs with the server, by including multiple TokenBinding structures in the TokenBindingMessage. By default, a client will establish a _provided_ Token Binding ID with the server, indicating a Token Binding ID that the client will persistently use with the server. Under certain conditions, the client can also include a _referred_ Token Binding ID in the TokenBindingMessage, indicating a Token Binding ID that the client is using with a _different_ server than the one that the TokenBindingMessage is sent to. This is useful in federation scenarios.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. The Sec-Token-Binding Header

Once a client and server have negotiated the Token Binding Protocol with HTTP/1.1 or HTTP/2 (see The Token Binding Protocol [TBPROTO]), clients MUST include the Sec-Token-Binding header in their HTTP requests. The ABNF of the Sec-Token-Binding header is:

```
Sec-Token-Binding = "Sec-Token-Binding" :: [CFWS] EncodedTokenBindingMessage
```

The EncodedTokenBindingMessage is a web-safe Base64-encoding of the TokenBindingMessage as defined in the TokenBindingProtocol [TBPROTO].

The TokenBindingMessage MUST contain a TokenBinding with TokenBindingType provided_token_binding, which MUST be signed with the Token Binding key used by the client for connections between itself and the server that the HTTP request is sent to (clients use...
different Token Binding keys for different servers). The Token Binding ID established by this TokenBinding is called a _Provided Token Binding ID_.

In HTTP/2, the client SHOULD use Header Compression [I-D.ietf-httpbis-header-compression] to avoid the overhead of repeating the same header in subsequent HTTP requests.

3. Federation Use Cases

3.1. Introduction

For privacy reasons, clients use different private keys to establish Provided Token Binding IDs with different servers. As a result, a server cannot bind a security token (such as an OAuth token or an OpenID Connect identity token) to a TLS connection that the client has with a different server. This is, however, a common requirement in federation scenarios: For example, an Identity Provider may wish to issue an identity token to a client and cryptographically bind that token to the TLS connection between the client and a Relying Party.

In this section we describe mechanisms to achieve this. The common idea among these mechanisms is that a server (called the _Token Consumer_ in this document) gives the client permission to reveal the Provided Token Binding ID that is used between the client and itself, to another server (called the _Token Provider_ in this document). Also common across the mechanisms is how the Token Binding ID is revealed to the Token Provider: The client uses the Token Binding Protocol [TBPROTO], and includes a TokenBinding structure in the Sec-Token-Binding HTTP header defined above. What differs between the various mechanisms is _how_ the Token Consumer grants the permission to reveal the Token Binding ID to the Token Provider. Below we specify one such mechanism, which is suitable for redirect-based interactions between Token Consumers and Token Providers.

3.2. Overview

In a Federated Sign-On protocol, an Identity Provider issues an identity token to a client, which sends the identity token to a Relying Party to authenticate itself. Examples of this include OpenID Connect (where the identity token is called "ID Token") and SAML (where the identity token is a SAML assertion).

To better protect the security of the identity token, the Identity Provider may wish to bind the identity token to the TLS connection between the client and the Relying Party, thus ensuring that only said client can use the identity token: The Relying Party will
compare the Token Binding ID in the identity token with the Token Binding ID of the TLS connection between it and the client.

This is an example of a federation scenario, which more generally can be described as follows:

- A Token Consumer causes the client to issue a token request to the Token Provider. The goal is for the client to obtain a token and then use it with the Token Consumer.

- The client delivers the token request to the Token Provider.

- The Token Provider issues the token. The token is issued for the specific Token Consumer who requested it (thus preventing malicious Token Consumers from using tokens with other Token Consumers). The token is, however, typically a bearer token, meaning that any client can use it with the Token Consumer, not just the client to which it was issued.

- Therefore, in the previous step, the Token Provider may want to include in the token the Token-Binding public key that the client uses when communicating with the Token Consumer, thus binding the token to client’s Token-Binding keypair. The client proves possession of the private key when communicating with the Token Consumer through the Token Binding Protocol [TBPROTO], and reveals the corresponding public key of this keypair as part of the Token Binding ID. Comparing the public key from the token with the public key from the Token Binding ID allows the Token Consumer to verify that the token was sent to it by the legitimate client.

- To allow the Token Provider to include the Token-Binding public key in the token, the Token Binding ID (between client and Token Consumer) must therefore be communicated to the Token Provider along with the token request. Communicating a Token Binding ID involves proving possession of a private key and is described in the Token Binding Protocol [TBPROTO].

The client will perform this last operation (proving possession of a private key that corresponds to a Token Binding ID between the client and the Token Consumer while delivering the token request to the Token Provider) only if the Token Consumer permits the client to do so.

Below, we specify how Token Consumers can grant this permission during redirect-based federation protocols.
3.3. HTTP Redirects

When a Token Consumer redirects the client to a Token Provider as a means to deliver the token request, it SHOULD include a Include-Referer-Token-Binding-ID HTTP response header in its HTTP response. The ABNF of the Include-Referer-Token-Binding-ID header is:

Include-Referer-Token-Binding-ID = "$Include-Referer-Token-Binding-ID\" :" [CFWS] %x74.72.75.65 ; "true", case-sensitive

Including this response header signals to the client that it should reveal the Token Binding ID used between the client and the Token Consumer to the Token Provider. In the absence of this response header, the client will not disclose any information about the Token Binding used between the client and the Token Consumer to the Token Provider.

This header has only meaning if the HTTP status code is 302 or 301, and MUST be ignored by the client for any other status codes. If the client supports the Token Binding Protocol, and has negotiated the Token Binding Protocol with both the Token Consumer and the Token Provider, it already sends the following header to the Token Provider with each HTTP request (see above):

Sec-Token-Binding: EncodedTokenBindingMessage

The TokenBindingMessage SHOULD contain a TokenBinding with TokenBindingType referred_token_binding. If included, this TokenBinding MUST be signed with the Token Binding key used by the client for connections between itself and the Token Consumer (more specifically, the web origin that issued the Include-Referer-Token-Binding-ID response header). The Token Binding ID established by this TokenBinding is called a _Referred Token Binding ID_.

As described above, the TokenBindingMessage MUST additionally contain a Provided Token Binding ID, i.e., a TokenBinding structure with TokenBindingType provided_token_binding, which MUST be signed with the Token Binding key used by the client for connections between itself and the Token Provider (more specifically, the web origin that the token request sent to).

3.4. Negotiated Key Parameters

The Token Binding Protocol [TBPROTO] allows the server and client to negotiate a signature algorithm used in the TokenBindingMessage. It is possible that the Token Binding ID used between the client and the
Token Consumer, and the Token Binding ID used between the client and Token Provider, use different signature algorithms. The client MUST use the signature algorithm negotiated with the Token Consumer in the referred_token_binding TokenBinding of the TokenBindingMessage, even if that signature algorithm is different from the one negotiated with the origin that the header is sent to.

Token Providers SHOULD support all the SignatureAndHashAlgorithms specified in the Token Binding Protocol [TBPROTO]. If a token provider does not support the SignatureAndHashAlgorithm specified in the referred_token_binding TokenBinding in the TokenBindingMessage, it MUST issue an unbound token.

4. Security Considerations

4.1. Security Token Replay

The goal of the Federated Token Binding mechanisms is to prevent attackers from exporting and replaying tokens used in protocols between the client and Token Consumer, thereby impersonating legitimate users and gaining access to protected resources. Bound tokens can still be replayed by malware present in the client. In order to export the token to another machine and successfully replay it, the attacker also needs to export the corresponding private key. The Token Binding private key is therefore a high-value asset and MUST be strongly protected, ideally by generating it in a hardware security module that prevents key export.

4.2. Privacy Considerations

The Token Binding protocol uses persistent, long-lived TLS Token Binding IDs. To protect privacy, TLS Token Binding IDs are never transmitted in clear text and can be reset by the user at any time, e.g. when clearing browser cookies. Unique Token Binding IDs MUST be generated for connections to different origins, so they cannot be used by cooperating servers to link user identities.

4.3. Triple Handshake Vulnerability in TLS

The Token Binding protocol relies on the tls_unique value to associate a TLS connection with a TLS Token Binding. The triple handshake attack [TRIPLE-HS] is a known TLS protocol vulnerability allowing the attacker to synchronize tls_unique values between TLS connections. The attacker can then successfully replay bound tokens. For this reason, the Token Binding protocol MUST NOT be negotiated unless the Extended Master Secret TLS extension [I-D.ietf-tls-session-hash] has also been negotiated.
5. References

5.1. Normative References

[I-D.ietf-httpbis-header-compression]


5.2. Informative References

[I-D.ietf-httpbis-http2]

[I-D.ietf-tls-session-hash]
[TRIPLE-HS]


Authors’ Addresses

Andrei Popov
Microsoft Corp.
USA

Email: andreipo@microsoft.com

Magnus Nystroem
Microsoft Corp.
USA

Email: mnystrom@microsoft.com

Dirk Balfanz (editor)
Google Inc.
USA

Email: balfanz@google.com

Adam Langley
Google Inc.
USA

Email: agl@google.com
The Token Binding Protocol Version 1.0
draft-ietf-tokbind-protocol-01

Abstract

This document specifies Version 1.0 of the Token Binding protocol. The Token Binding protocol allows client/server applications to create long-lived, uniquely identifiable TLS [RFC5246] bindings spanning multiple TLS sessions and connections. Applications are then enabled to cryptographically bind security tokens to the TLS layer, preventing token export and replay attacks. To protect privacy, the TLS Token Binding identifiers are only transmitted encrypted and can be reset by the user at any time.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on November 30, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of production of this document.
1. Introduction

Web services generate various security tokens (e.g. HTTP cookies, OAuth tokens) for web applications to access protected resources. Any party in possession of such token gains access to the protected resource. Attackers export bearer tokens from the user’s machine, present them to web services, and impersonate authenticated users. The idea of Token Binding is to prevent such attacks by cryptographically binding security tokens to the TLS layer.

A TLS Token Binding is established by the user agent generating a private-public key pair (possibly within a secure hardware module, such as TPM) per target server, and proving possession of the private key on every TLS connection to the target server. The proof of possession involves signing the tls_unique value [RFC5929] for the TLS connection with the private key. Such TLS Token Binding is identified by the corresponding public key. TLS Token Bindings are long-lived, i.e. they encompass multiple TLS connections and TLS sessions between a given client and server. To protect privacy, TLS
Token Binding identifiers are never transmitted in clear text and can be reset by the user at any time, e.g. when clearing browser cookies.

When issuing a security token to a client that supports TLS Token Binding, a server includes the client’s TLS Token Binding ID in the token. Later on, when a client presents a security token containing a TLS Token Binding ID, the server makes sure the ID in the token matches the ID of the TLS Token Binding established with the client. In the case of a mismatch, the server discards the token.

In order to successfully export and replay a bound security token, the attacker needs to also be able to export the client’s private key, which is hard to do in the case of the key generated in a secure hardware module.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Token Binding Protocol Overview

The client and server use the Token Binding Negotiation TLS Extension [TBNEGO] to negotiate the Token Binding protocol version and the parameters (signature and hash algorithm, length) of the Token Binding key. This negotiation does not require additional round-trips.

The Token Binding protocol consists of one message sent by the client to the server, proving possession of one or more client-generated asymmetric keys. This message is only sent if the client and server agree on the use of the Token Binding protocol and the key parameters. The Token Binding message is sent with the application protocol data in TLS application_data records.

A server receiving the Token Binding message verifies that the key parameters in the message match the Token Binding parameters negotiated via [TBNEGO], and then validates the signatures contained in the Token Binding message. If either of these checks fails, the server terminates the connection, otherwise the TLS Token Binding is successfully established with the ID contained in the Token Binding message.

When a server supporting the Token Binding protocol receives a bound token, the server compares the TLS Token Binding ID in the security token with the TLS Token Binding ID established with the client. If
the bound token came from a TLS connection without a Token Binding, or if the IDs don’t match, the token is discarded.

This document defines the format of the Token Binding protocol message, the process of establishing a TLS Token Binding, the format of the Token Binding ID, and the process of validating a security token. Token Binding Negotiation TLS Extension [TBNEGO] describes the negotiation of the Token Binding protocol and key parameters. Token Binding over HTTP [HTTPSTB] explains how the Token Binding message is encapsulated within HTTP/1.1 [RFC7230] or HTTP/2 [RFC7540] messages. [HTTPSTB] also describes Token Binding between multiple communicating parties: User Agent, Identity Provider and Relying Party.

3. Token Binding Protocol Message

The Token Binding message is sent by the client and proves possession of one or more private keys held by the client. This message MUST be sent if the client and server successfully negotiated the use of the Token Binding protocol via [TBNEGO], and MUST NOT be sent otherwise. This message MUST be sent in the client’s first application protocol message. This message MAY also be sent in subsequent application protocol messages, proving possession of other keys by the same client, to facilitate token binding between more than two communicating parties. Token Binding over HTTP [HTTPSTB] specifies the encapsulation of the Token Binding message in the application protocol messages, and the scenarios involving more than two communicating parties. The Token Binding message format is defined using TLS specification language, and reuses existing TLS structures and IANA registrations where possible:

```c
enum {
    sha256(4), (255)
} HashAlgorithm;

enum {
    rsa(1), ecdsap256(3), (255)
} SignatureAlgorithm;

struct {
    HashAlgorithm hash;
    SignatureAlgorithm signature;
} SignatureAndHashAlgorithm;

struct {
    opaque modulus<1..2^16-1>;
    opaque publicexponent<1..2^8-1>;
}
```
enum {
    secp256r1 (23), (0xFFFF)
} NamedCurve;

struct {
    opaque point <1..2^8-1>;
} ECPoint;

struct {
    NamedCurve namedcurve;
    ECPoint point;              // Uncompressed format
} ECDSAParams;

enum {
    provided_token_binding(0), referred_token_binding(1), (255)
} TokenBindingType;

struct {
    TokenBindingType tokenbinding_type;
    SignatureAndHashAlgorithm algorithm;
    select (algorithm.signature) {
        case rsa: RSAPublicKey rsapubkey;
        case ecdsa: ECDSAParams ecdsaparams;
    }
} TokenBindingID;

enum {
    (255)                       // No initial ExtensionType registrations
} ExtensionType;

struct {
    ExtensionType extension_type;
    opaque extension_data<0..2^16-1>;
} Extension;

struct {
    TokenBindingID tokenbindingid;
    opaque signature<0..2^16-1>; // Signature over hashed ("token binding", tls_unique)
    Extension extensions<0..2^16-1>;
} TokenBinding;

struct {
    TokenBinding tokenbindings<0..2^16-1>;
} TokenBindingMessage;
The Token Binding message consists of a series of TokenBinding structures containing the TokenBindingID, a signature over the hash of the NUL-terminated, ASCII label ("token binding") and the tls_unique, optionally followed by Extension structures. An implementation MUST ignore any unknown extensions. Initially, no extension types are defined. At least one TokenBinding MUST be included in the Token Binding message. The signature and hash algorithm and key length used in the TokenBinding MUST match the parameters negotiated via [TBNEGO]. The client SHOULD generate and store Token Binding keys in a secure manner that prevents key export. In order to prevent cooperating servers from linking user identities, different keys SHOULD be used by the client for connections to different servers, according to the token scoping rules of the application protocol.

4. Establishing a TLS Token Binding

The triple handshake vulnerability in the TLS protocol affects the security of the Token Binding protocol, as described in the "Security Considerations" section below. Therefore, the server MUST NOT negotiate the use of the Token Binding protocol unless the server also negotiates Extended Master Secret TLS extension [I-D.ietf-tls-session-hash].

The server MUST terminate the connection if the use of the Token Binding protocol has been successfully negotiated via [TBNEGO] within the TLS handshake, but the client’s first application message does not contain the Token Binding message. The server MUST terminate the connection if the use of the Token Binding protocol was not negotiated, but the client sends the Token Binding message.

If the Token Binding type is "provided_token_binding", the server MUST verify that the signature algorithm (including elliptic curve in the case of ECDSA) and key length in the Token Binding message match those negotiated via [TBNEGO]. In the case of a mismatch, the server MUST terminate the connection. As described in [HTTPSTB], Token Bindings of type "referred_token_binding" may have different key parameters than those negotiated via [TBNEGO].

If the Token Binding message does not contain at least one TokenBinding structure, or the signature contained in a TokenBinding structure is invalid, the server MUST terminate the connection. Otherwise, the TLS Token Binding is successfully established and its ID can be provided to the application for security token validation.
5. TLS Token Binding ID Format

The ID of the TLS Token Binding established as a result of Token Binding message processing is a binary representation of the following structure:

```c
struct {
    TokenBindingType tokenbinding_type;
    SignatureAndHashAlgorithm algorithm;
    select (algorithm.signature) {
        case rsa: RSAPublicKey rsapubkey;
        case ecdsa: ECDSAParams ecdsaparams;
    }
} TokenBindingID;
```

TokenBindingID includes the type of the token binding and the key parameters negotiated via [TBNEGO]. This document defines two token binding types: provided_token_binding used to establish a Token Binding when connecting to a server, and referred_token_binding used when requesting tokens to be presented to a different server. Token Binding over HTTP [HTTPSTB] describes Token Binding between multiple communicating parties: User Agent, Identity Provider and Relying Party. TLS Token Binding ID can be obtained from the TokenBinding structure described in the "Token Binding Protocol Message" section of this document by discarding the signature and extensions. TLS Token Binding ID will be available at the application layer and used by the server to generate and verify bound tokens.

6. Security Token Validation

Security tokens can be bound to the TLS layer either by embedding the Token Binding ID in the token, or by maintaining a database mapping tokens to Token Binding IDs. The specific method of generating bound security tokens is application-defined and beyond the scope of this document.

Upon receipt of a security token, the server attempts to retrieve TLS Token Binding ID information from the token and from the TLS connection with the client. Application-provided policy determines whether to honor non-bound (bearer) tokens. If the token is bound and a TLS Token Binding has not been established for the client connection, the server MUST discard the token. If the TLS Token Binding ID for the token does not match the TLS Token Binding ID established for the client connection, the server MUST discard the token.
7. IANA Considerations

This document establishes a registry for Token Binding type identifiers entitled "Token Binding Types" under the "Token Binding Protocol" heading.

Entries in this registry require the following fields:

- **Value**: The octet value that identifies the Token Binding type (0-255).
- **Description**: The description of the Token Binding type.
- **Specification**: A reference to a specification that defines the Token Binding type.

This registry operates under the "Expert Review" policy as defined in [RFC5226]. The designated expert is advised to encourage the inclusion of a reference to a permanent and readily available specification that enables the creation of interoperable implementations using the identified Token Binding type.

An initial set of registrations for this registry follows:

- **Value**: 0
  - **Description**: provided_token_binding
  - **Specification**: this document

- **Value**: 1
  - **Description**: referred_token_binding
  - **Specification**: this document

This document establishes a registry for Token Binding extensions entitled "Token Binding Extensions" under the "Token Binding Protocol" heading.

Entries in this registry require the following fields:

- **Value**: The octet value that identifies the Token Binding extension (0-255).
- **Description**: The description of the Token Binding extension.
8. Security Considerations

8.1. Security Token Replay

The goal of the Token Binding protocol is to prevent attackers from exporting and replaying security tokens, thereby impersonating legitimate users and gaining access to protected resources. Bound tokens can still be replayed by the malware present in the User Agent. In order to export the token to another machine and successfully replay it, the attacker also needs to export the corresponding private key. Token Binding private keys are therefore high-value assets and SHOULD be strongly protected, ideally by generating them in a hardware security module that prevents key export.

8.2. Downgrade Attacks

The Token Binding protocol is only used when negotiated via [TBNEGO] within the TLS handshake. TLS prevents active attackers from modifying the messages of the TLS handshake, therefore it is not possible for the attacker to remove or modify the Token Binding Negotiation TLS Extension used to negotiate the Token Binding protocol and key parameters. The signature algorithm and key length used in the TokenBinding of type "provided_token_binding" MUST match the parameters negotiated via [TBNEGO].

8.3. Privacy Considerations

The Token Binding protocol uses persistent, long-lived TLS Token Binding IDs. To protect privacy, TLS Token Binding IDs are never transmitted in clear text and can be reset by the user at any time, e.g. when clearing browser cookies. In order to prevent cooperating
servers from linking user identities, different keys SHOULD be used by the client for connections to different servers, according to the token scoping rules of the application protocol.

8.4. Token Binding Key Sharing Between Applications

Existing systems provide a variety of platform-specific mechanisms for certain applications to share tokens, e.g. to enable single sign-on scenarios. For these scenarios to keep working with bound tokens, the applications that are allowed to share tokens will need to also share Token Binding keys. Care must be taken to restrict the sharing of Token Binding keys to the same group(s) of applications that share the same tokens.

8.5. Triple Handshake Vulnerability in TLS

The Token Binding protocol relies on the tls_unique value to associate a TLS connection with a TLS Token Binding. The triple handshake attack [TRIPLE-HS] is a known TLS protocol vulnerability allowing the attacker to synchronize tls_unique values between TLS connections. The attacker can then successfully replay bound tokens. For this reason, the Token Binding protocol MUST NOT be negotiated unless the Extended Master Secret TLS extension [I-D.ietf-tls-session-hash] has also been negotiated.

9. References

9.1. Normative References


9.2. Informative References

[I-D.ietf-tls-session-hash]

[TRIPLE-HS]

Authors' Addresses

Andrei Popov (editor)
Microsoft Corp.
USA
Email: andreip@microsoft.com

Magnus Nystroem
Microsoft Corp.
USA
Email: mnystrom@microsoft.com

Dirk Balfanz
Google Inc.
USA
Email: balfanz@google.com
Abstract

This document specifies a Transport Layer Security (TLS) [RFC5246] extension for the negotiation of Token Binding protocol [TBPROTO] version and key parameters.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on November 30, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of
1. Introduction

In order to use the Token Binding protocol [TBPROTO], the client and server need to agree on the Token Binding protocol version and the parameters (signature and hash algorithm, length) of the Token Binding key. This document specifies a new TLS extension to accomplish this negotiation without introducing additional network round-trips.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Token Binding Negotiation Client Hello Extension

The client uses the "token_binding" TLS extension to indicate supported Token Binding protocol version and key parameters.

```c
enum {
    token_binding(TBD), (65535)
} ExtensionType;
```

The "extension_data" field of this extension contains a "TokenBindingParameters" value.
struct {
    uint8 major;
    uint8 minor;
} ProtocolVersion;

enum {
    rsa2048_pkcs1.5_sha256(0), rsa2048_pss_sha256(1), ecdsap256_sha256(2), (255)
} TokenBindingKeyParameters;

struct {
    ProtocolVersion token_binding_version;
    TokenBindingKeyParameters key_parameters_list<2..2^16-1>
} TokenBindingParameters;

"token_binding_version" indicates the supported version of the Token Binding protocol. [TBPROTO] describes version {1, 0} of the protocol. Prototype implementations of Token Binding drafts can indicate support of a specific draft version, e.g. {0, 0} or {0, 1}.

"key_parameters_list" contains the list of identifiers of the Token Binding key parameters supported by the client, in descending order of preference.

3. Token Binding Negotiation Server Hello Extension

The server uses the "token_binding" TLS extension to indicate support for the Token Binding protocol version offered by the client and to select key parameters.

The server that supports Token Binding and receives a client hello message containing the "token_binding" extension, will include the "token_binding" extension in the server hello if all of the following conditions are satisfied:

1. The server supports the Token Binding protocol version offered by the client.
2. The server finds acceptable Token Binding key parameters on the client’s list.
3. The server is also negotiating Extended Master Secret TLS extension [I-D.ietf-tls-session-hash] (see security considerations section below for more details).

The server will ignore any key parameters that it does not recognize.

The "extension_data" field of the "token_binding" extension is structured the same as described above for the client "extension_data".
"token_binding_version" echos the Token Binding protocol version advertised by the client.

"key_parameters_list" contains exactly one Token Binding key parameters identifier selected by the server from the client’s list.

4. Negotiating Token Binding Protocol Version and Key Parameters

It is expected that a server will have a list of Token Binding key parameters identifiers that it supports, in preference order. The server MUST only select an identifier that the client offered. The server SHOULD select the most highly preferred key parameters identifier it supports which is also advertised by the client. In the event that the server supports none of the key parameters that the client advertises, then the server MUST NOT include "token_binding" extension in the server hello.

The client receiving the "token_binding" extension MUST terminate the handshake with a fatal "unsupported_extension" alert if any of the following conditions are true:

1. The client did not include the "token_binding" extension in the client hello.

2. "token_binding_version" does not match the Token Binding protocol version advertised by the client.

3. "key_parameters_list" includes more than one Token Binding key parameters identifier.

4. "key_parameters_list" includes an identifier that was not advertised by the client.

5. Extended Master Secret [I-D.ietf-tls-session-hash] is not negotiated (see security considerations section below for more details).

If the "token_binding" extension is included in the server hello and the TLS handshake succeeds, it means that the Token Binding protocol version and key parameters have been negotiated between the client and the server and SHALL be definitive for the TLS session. In this case, the client MUST use the negotiated key parameters in the "provided_token_binding" as described in [TBPROTO].
5. IANA Considerations

This document defines a new TLS extension "token_binding", which needs to be added to the IANA "Transport Layer Security (TLS) Extensions" registry.

This document establishes a registry for identifiers of Token Binding key parameters entitled "Token Binding Key Parameters" under the "Token Binding Protocol" heading.

Entries in this registry require the following fields:

- **Value**: The octet value that identifies a set of Token Binding key parameters (0-255).
- **Description**: The description of the Token Binding key parameters.
- **Specification**: A reference to a specification that defines the Token Binding key parameters.

This registry operates under the "Expert Review" policy as defined in [RFC5226]. The designated expert is advised to encourage the inclusion of a reference to a permanent and readily available specification that enables the creation of interoperable implementations using the identified set of Token Binding key parameters.

An initial set of registrations for this registry follows:

- **Value**: 0
  - **Description**: rsa2048_pkcs1_5_sha256
  - **Specification**: this document

- **Value**: 1
  - **Description**: rsa2048_pss_sha256
  - **Specification**: this document

- **Value**: 2
  - **Description**: ecdsap256_sha256
  - **Specification**: this document
6. Security Considerations

6.1. Downgrade Attacks

The Token Binding protocol version and key parameters are negotiated via "token_binding" extension within the TLS handshake. TLS prevents active attackers from modifying the messages of the TLS handshake, therefore it is not possible for the attacker to remove or modify the "token_binding" extension. The signature and hash algorithms and key length used in the TokenBinding of type "provided_token_binding" MUST match the parameters negotiated via "token_binding" extension.

6.2. Triple Handshake Vulnerability in TLS

The Token Binding protocol relies on the tls_unique value to associate a TLS connection with a TLS Token Binding. The triple handshake attack [TRIPLE-HS] is a known TLS protocol vulnerability allowing the attacker to synchronize tls_unique values between TLS connections. The attacker can then successfully replay bound tokens. For this reason, the Token Binding protocol MUST NOT be negotiated unless the Extended Master Secret TLS extension [I-D.ietf-tls-session-hash] has also been negotiated.

7. Acknowledgements

This document incorporates comments and suggestions offered by Eric Rescorla, Gabriel Montenegro, Martin Thomson, Vinod Anupam.

8. References

8.1. Normative References


8.2. Informative References

[I-D.ietf-tls-session-hash]

[TRIPLE-HS]

Authors’ Addresses

Andrei Popov (editor)
Microsoft Corp.
USA
Email: andreipo@microsoft.com

Magnus Nystroem
Microsoft Corp.
USA
Email: mnystrom@microsoft.com

Dirk Balfanz
Google Inc.
USA
Email: balfanz@google.com

Adam Langley
Google Inc.
USA
Email: agl@google.com